# Robotics

## L09: Motion Planning

Dr. Mohammed Elmogy

Mansoura University
Faculty of Computers & Information Sciences
Department of Information Systems

**elmogy@gmail.com**

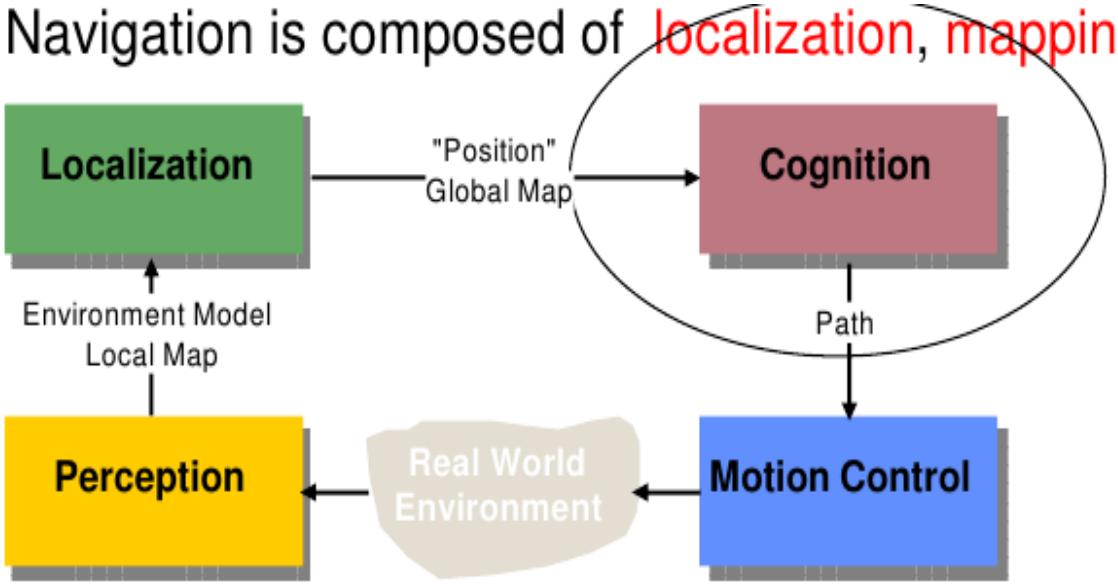
17. December 2011

# Outline

# Outline

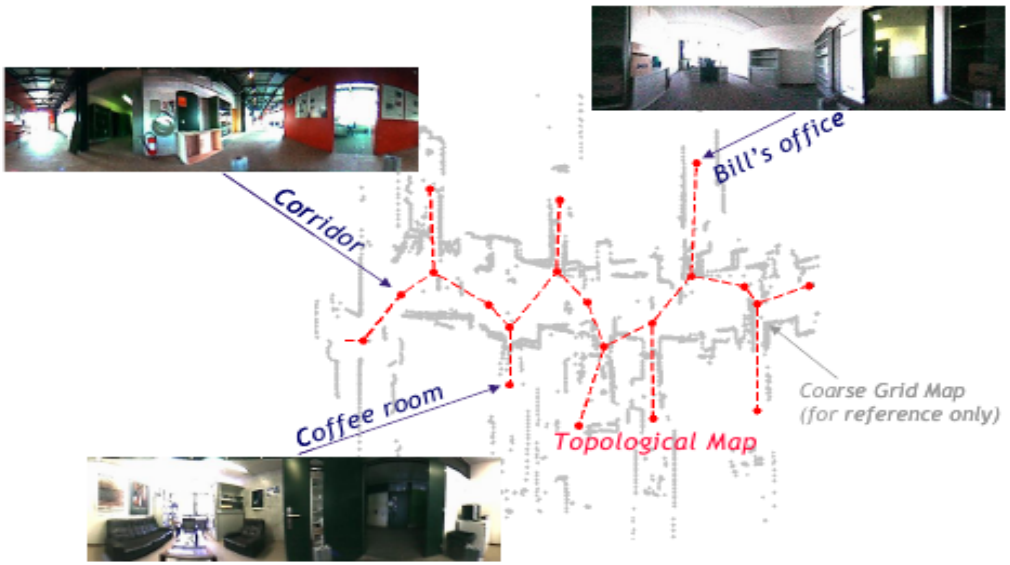
# Required Competences for Navigation

- Navigation is composed of localization, mapping and motion planning



# The Planning Problem

- The problem: find a path in the work space (physical space) from an initial position to a goal position avoiding all collisions with obstacles
- Assumption: there exists a good enough map of the environment for navigation.
  - Topological
  - Metric
  - Hybrid methods

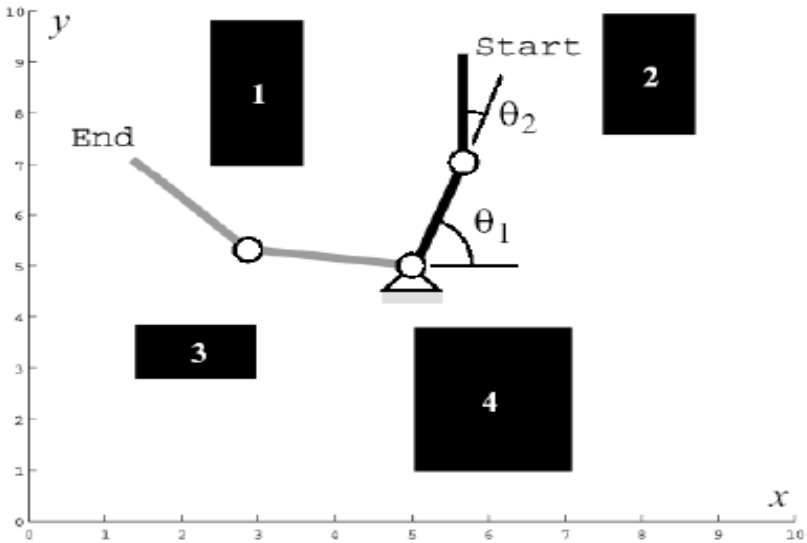


# The Planning Problem (cont.)

- We can generally distinguish between
  - (global) path planning and
  - (local) obstacle avoidance.
- First step:
  - Transformation of the map into a representation useful for planning
  - This step is planner-dependent
- Second step:
  - Plan a path on the transformed map
- Third step:
  - Send motion commands to controller
  - This step is planner-dependent (e.g. Model based feed forward, path following)

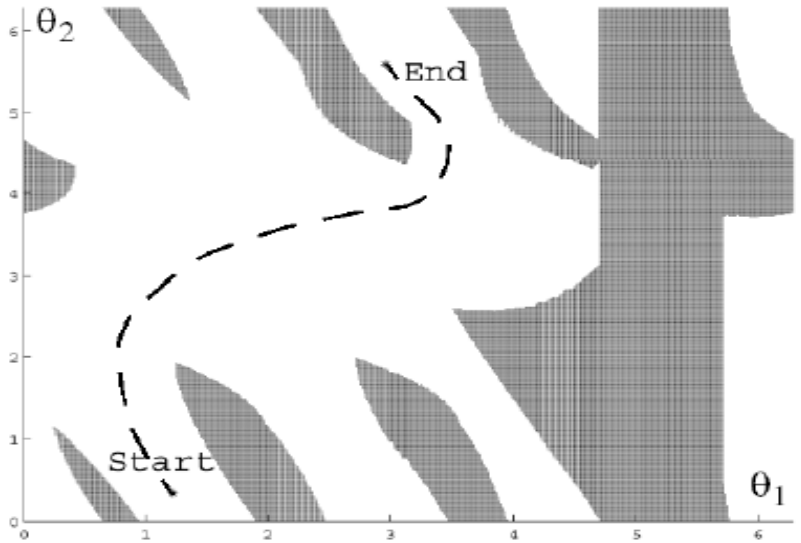
# Work Space (Map) $\Longrightarrow$ Configuration Space

- State or configuration $q$ can be described with $k$ values $q_i$




a)

**Work Space**

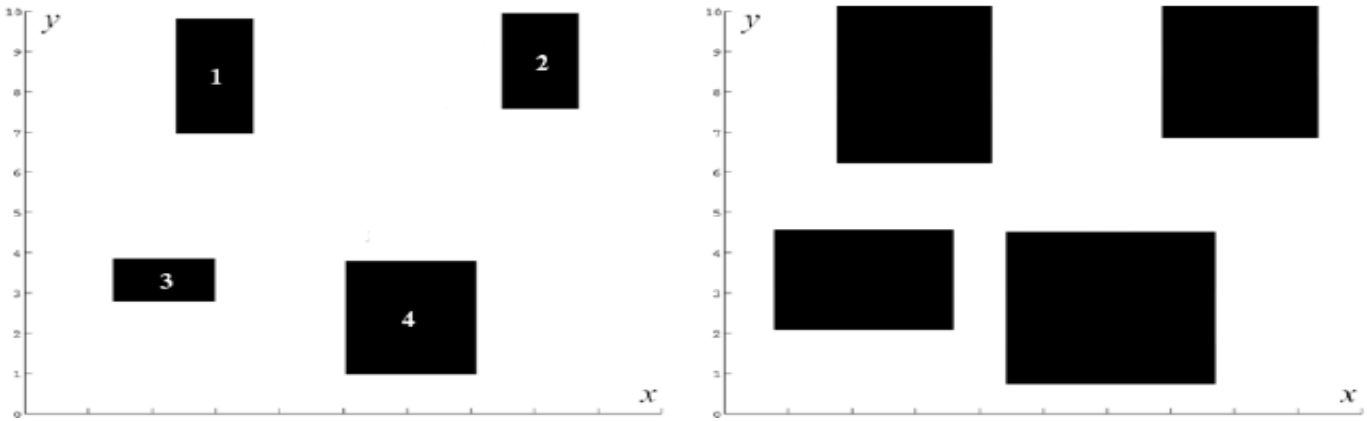



b)

**Configuration Space:**
the dimension of this space is equal to the Degrees of Freedom (DoF) of the robot

# Configuration Space for a Mobile Robot

- Mobile robots operating on a flat ground have 3 DoF: (x, y, θ)
- For simplification, in path planning mobile roboticists often assume that the robot is holonomic and that it is a point. In this way the configuration space is reduced to 2D (x,y)
- Because we have reduced each robot to a point, we have to inflate each obstacle by the size of the robot radius to compensate.

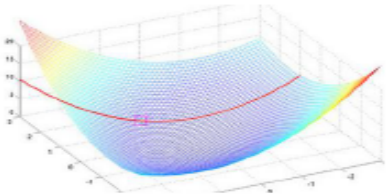


# Outline

# Path Planning: Overview of Algorithms

## 1. Optimal Control

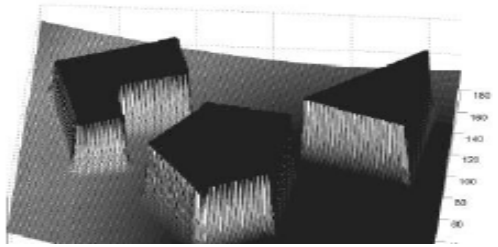
- Solves for the truly optimal solution
- Becomes intractable for even moderately complex and/or nonconvex problems



Source:
http://mitocw.udsm.ac.tz

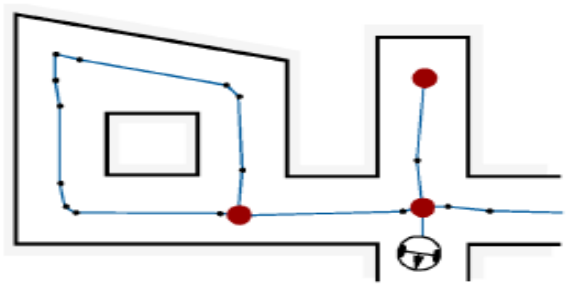
## 2. Potential Field

- Imposes a mathematical function over the state/configuration space
- Many physical metaphors exist
- Often employed due to its simplicity and similarity to optimal control solutions

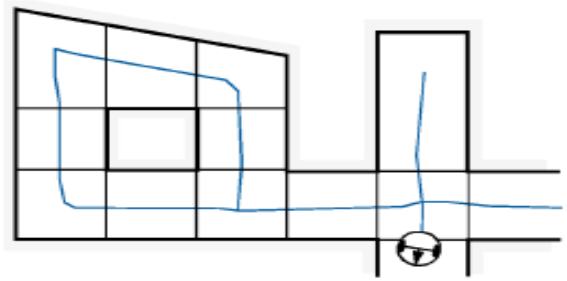


## 3. Graph Search

- Identify a set of edges and connect them to nodes within the free space

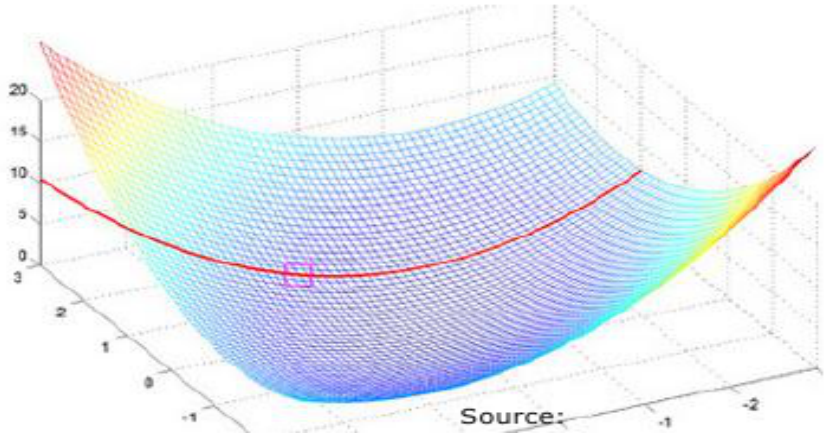


- Where to put the nodes?

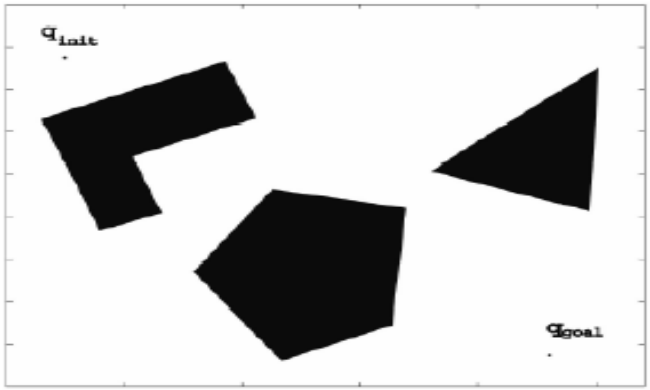


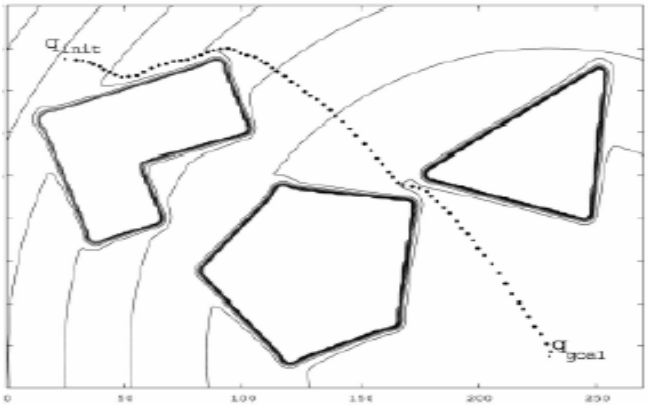
# Optimal Control based Path Planning Strategies

- Overview
  - Solves a two-point boundary problem in the continuum
- Limitations
  - Becomes very hard to solve as problem dimensionality increases
  - Prone to local optima
- Algorithms
  - Pontryagin maximum principle
  - Hamilton-Jacobi-Bellman

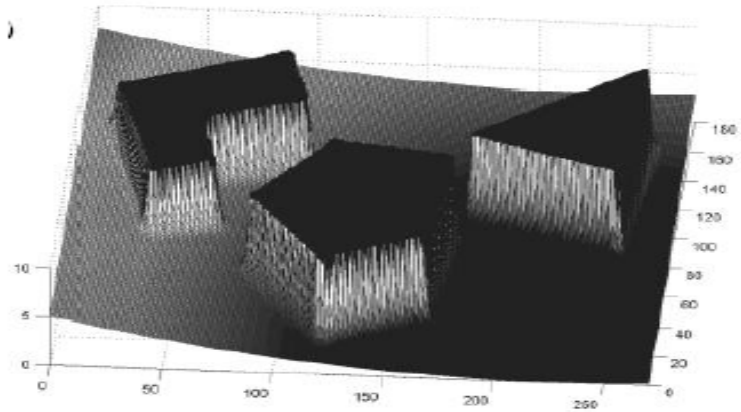


# Potential Field Path Planning Strategies




- Robot is treated as a *point under the influence* of an artificial potential field.
- Operates in the continuum
  - Generated robot movement is similar to a ball rolling down the hill
  - Goal generates attractive force
  - Obstacle are repulsive forces



# Graph Search

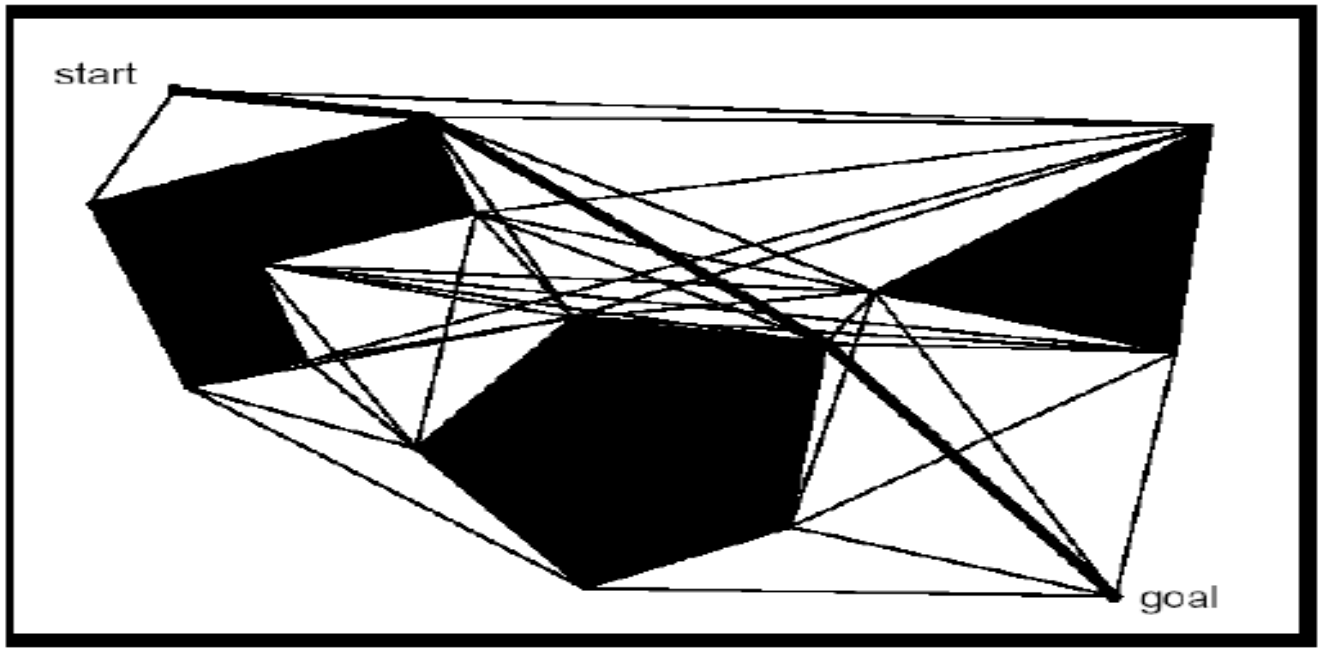
- Overview
  - Solves a least cost problem between two states on a (directed) graph
  - Graph structure is a discrete representation
- Limitations
  - State space is discretized → completeness is at stake
  - Feasibility of paths is often not inherently encoded
- Algorithms
  - (Preprocessing steps)
  - Breath first
  - Depth first
  - Dijkstra
  - A* and variants
  - D* and variants

# Graph Construction (Preprocessing Step)

**Methods:**

- Visibility graph
- Voronoi diagram
- Cell decomposition
- .....

# Graph Construction: Visibility Graph

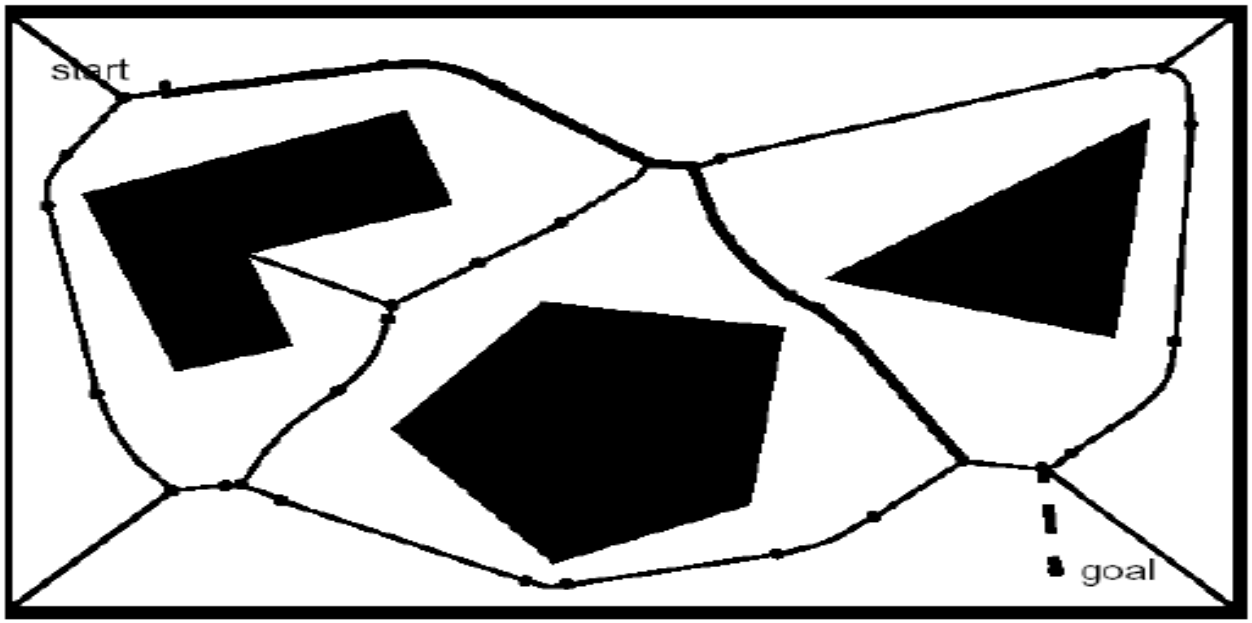



- Particularly suitable for polygon-like obstacles
- Shortest path length
- Grow obstacles to avoid collisions

# Graph Construction: Visibility Graph (cont.)

- Pros
  - The found path is optimal because it is the shortest length path
  - Implementation simple when obstacles are polygons
- Cons
  - The solution path found by the visibility graph tend to take the robot as close as possible to the obstacles: the common solution is to grow obstacles by more than robot's radius
  - Number of edges and nodes increases with the number of polygons
  - Thus it can be inefficient in densely populated environments

# Graph Construction: Voronoi Diagram



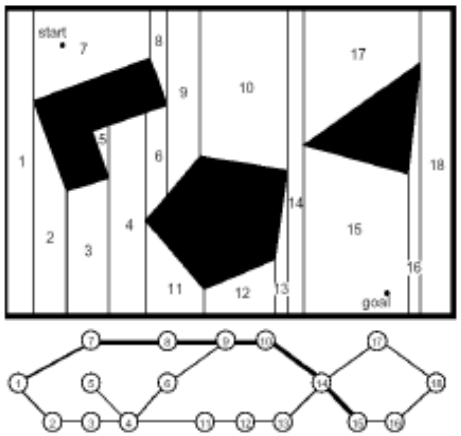

- In contrast to the Visibility Graph approach, the Voronoi Diagram tends to maximize the distance between robot and obstacles
- Easily executable: Maximize the sensor readings
- Works also for map-building: Move on the Voronoi edges: 1D Mapping

# Graph Construction: Voronoi Diagram (cont.)

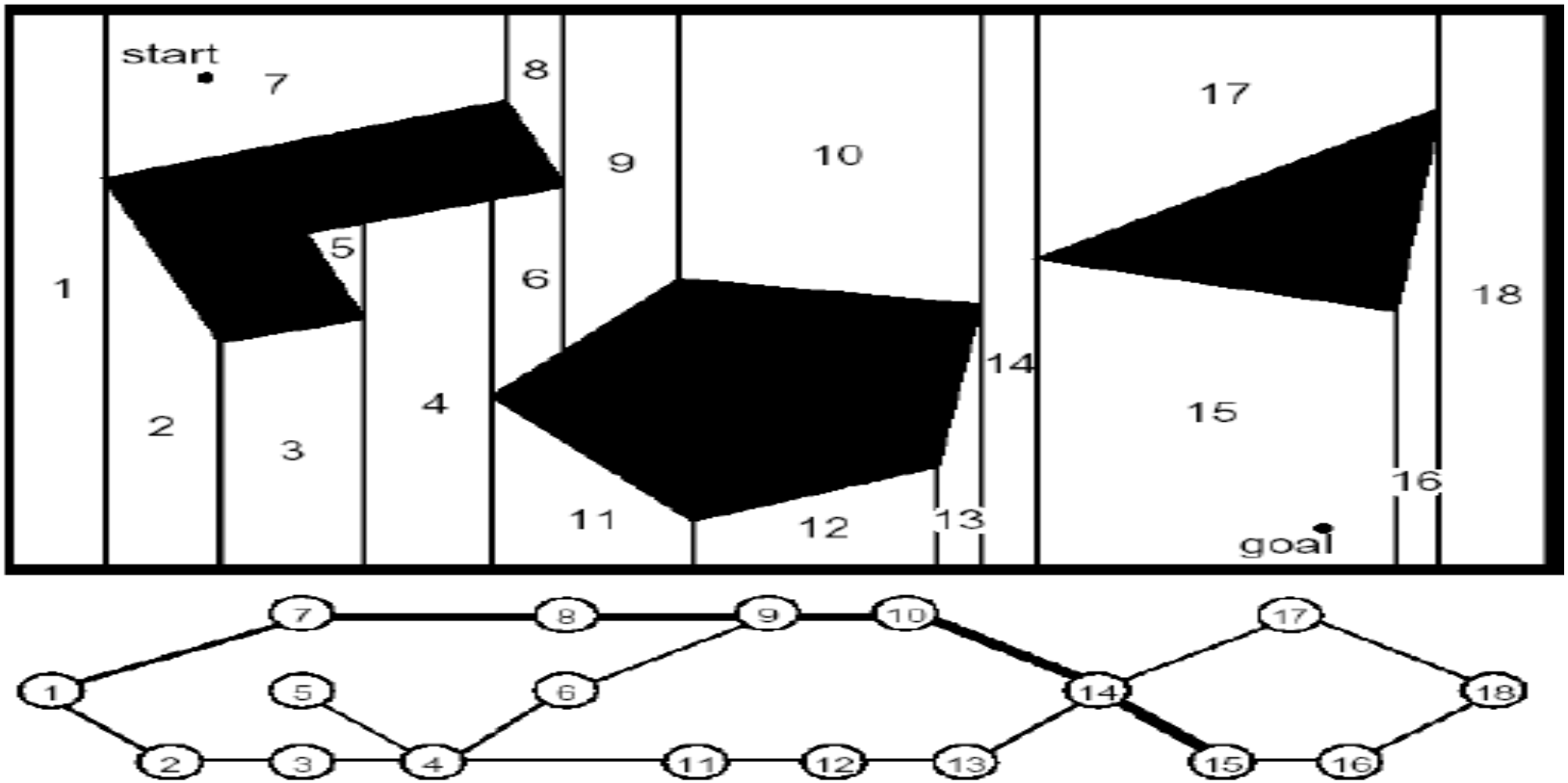
- Pros
  - Using range sensors like laser or sonar, a robot can navigate along the Voronoi diagram using simple control rules
- Cons
  - Because the Voronoi diagram tends to keep the robot as far as possible from obstacles, any short range sensor will be in danger of failing
- Peculiarities
  - when obstacles are polygons, the Voronoi map consists of straight and parabolic segments

# Graph Construction: Cell Decomposition

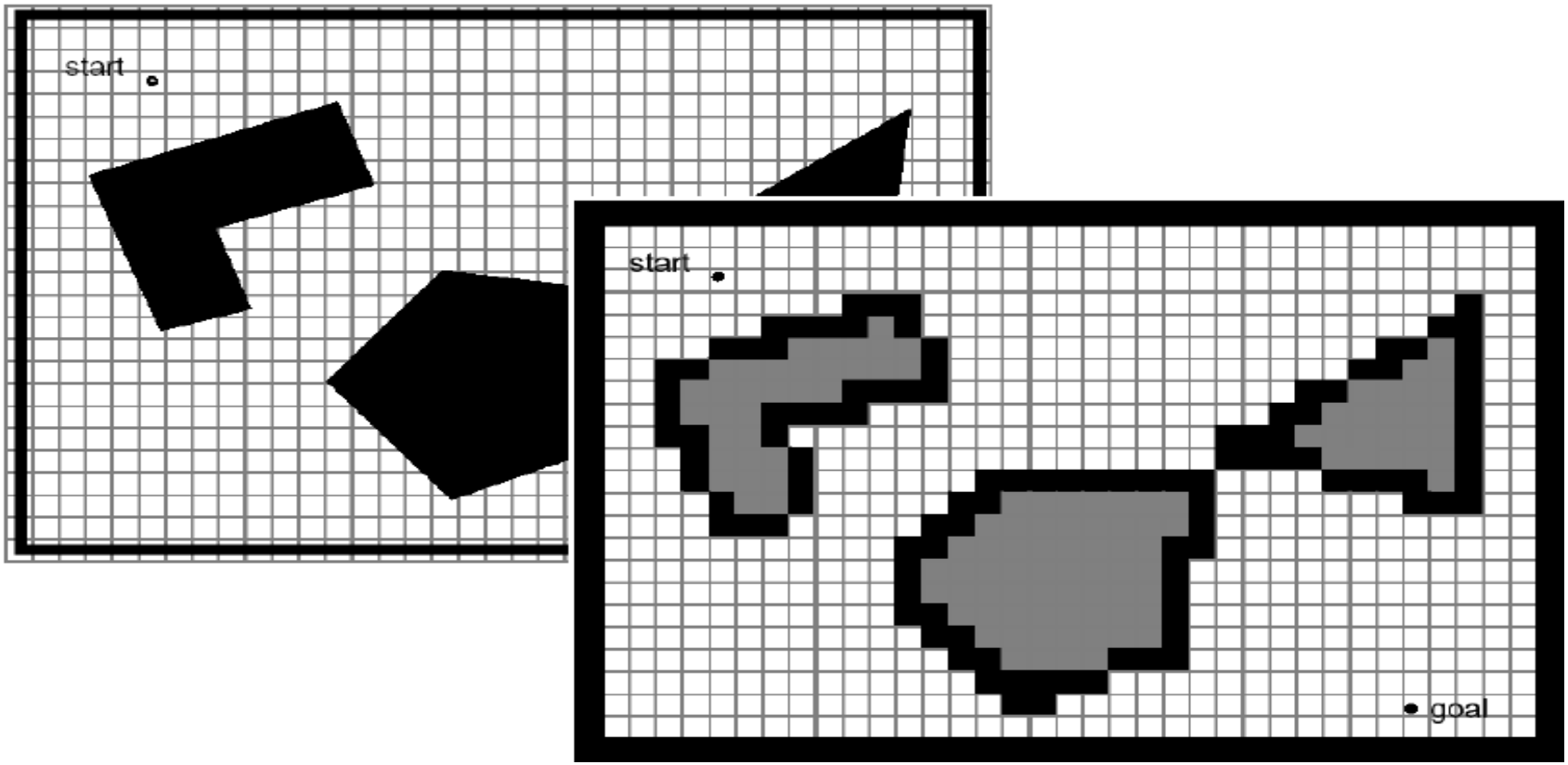
- Divide space into simple, connected regions called cells
- Determine which open cells are adjacent and construct a connectivity graph
- Find cells in which the initial and goal configuration (state) lie and search for a path in the connectivity graph to join them.
- From the sequence of cells found with an appropriate search algorithm, compute a path within each cell.
  - e.g. passing through the midpoints of cell boundaries or by sequence of wall following movements.
- Possible cell decompositions:
  - Exact cell decomposition
  - Approximate cell decomposition:
    - Fixed cell decomposition
    - Adaptive cell decomposition



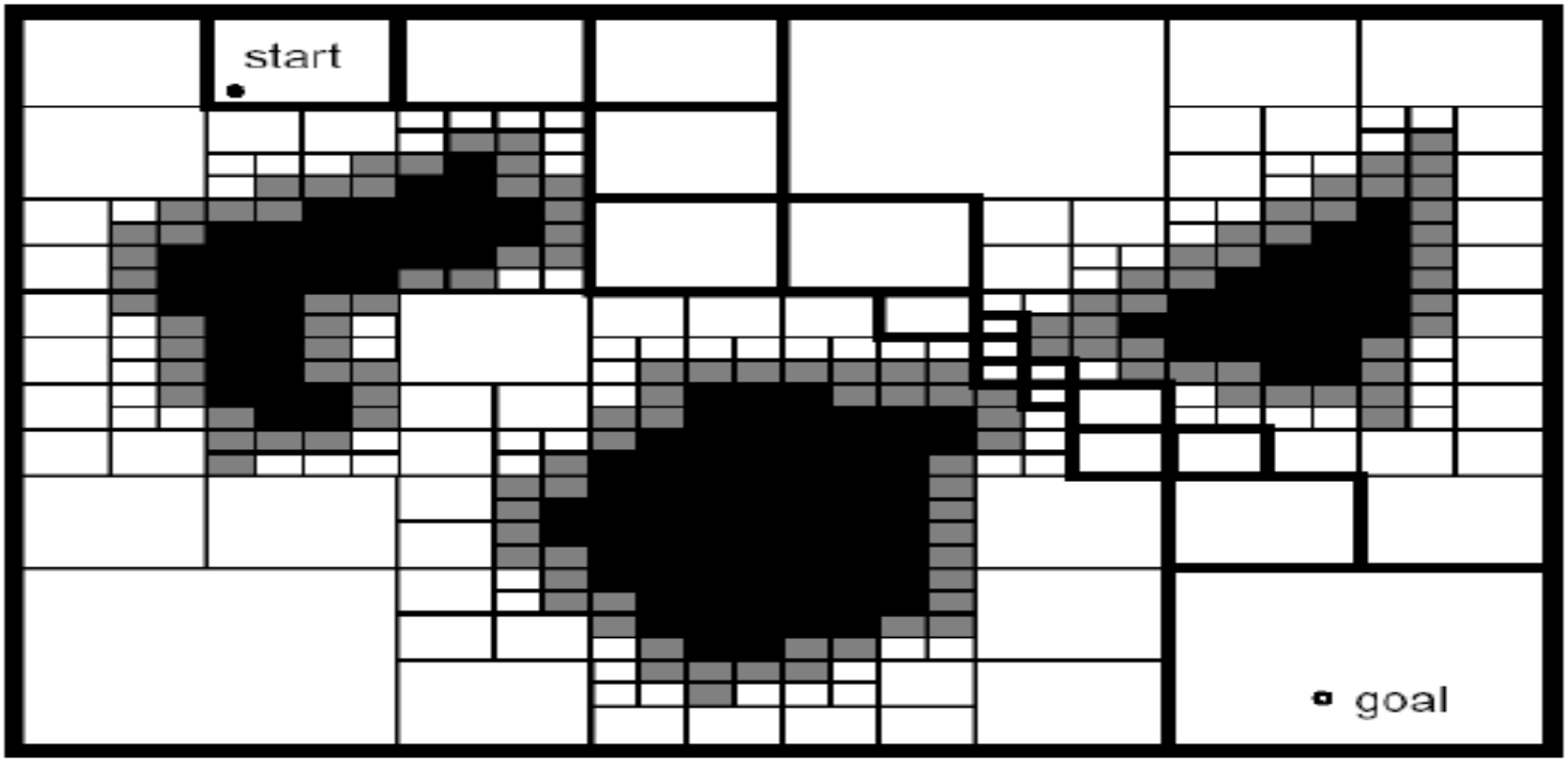
# Graph Construction: Cell Decomposition (cont.)



# Graph Construction: Cell Decomposition (cont.)

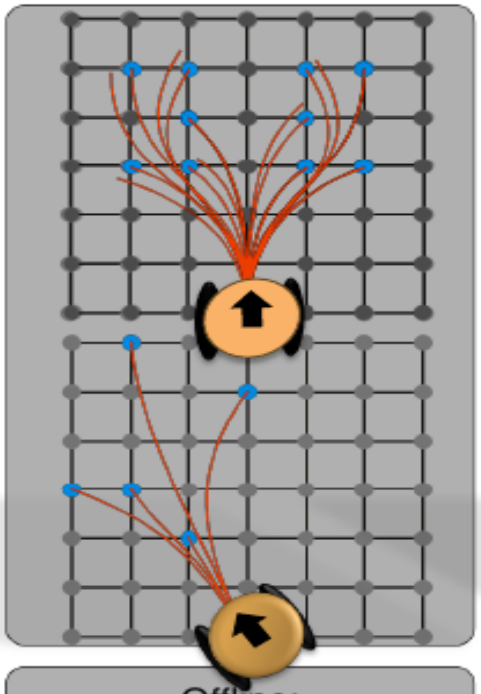


# Graph Construction: Cell Decomposition (cont.)

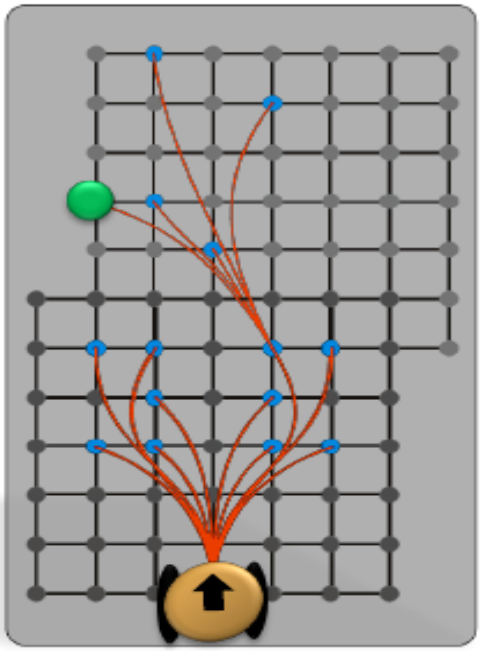


# Graph Construction: State Lattice Design

Offline:
Motion Model



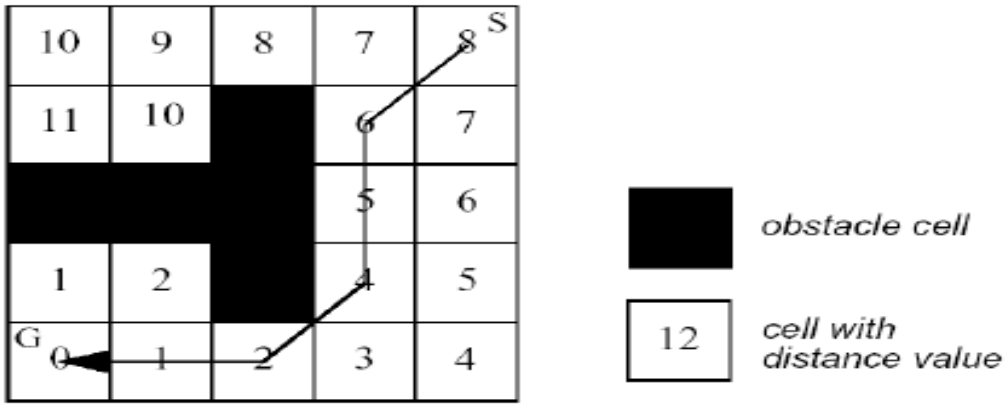
Offline:
Lattice Gen.



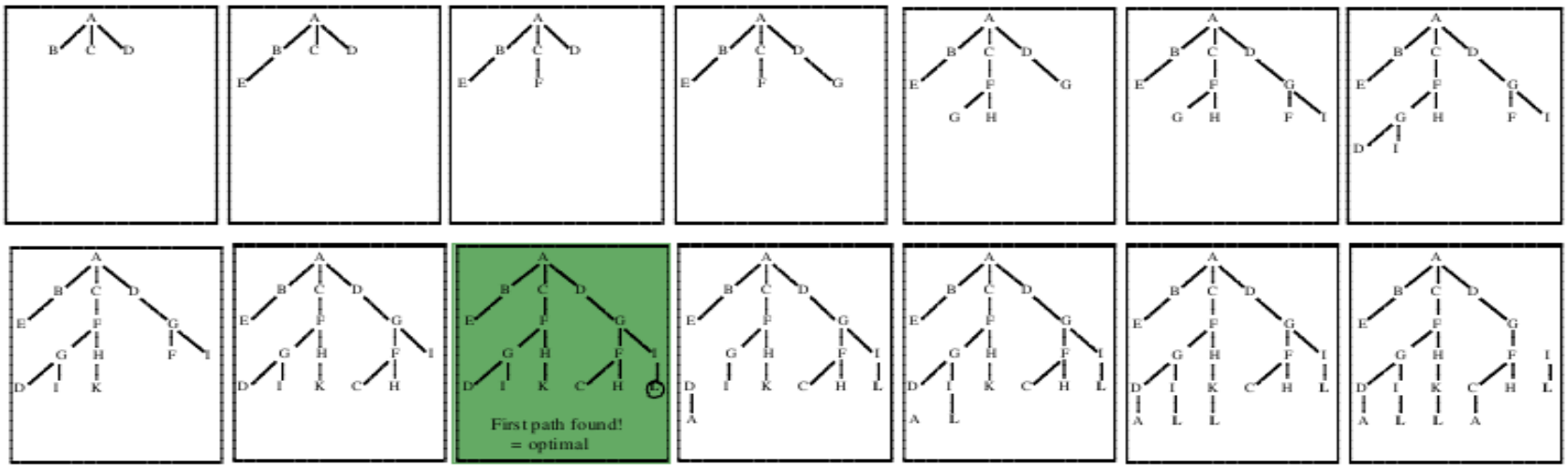
Online:
Incremental Graph
Constr.

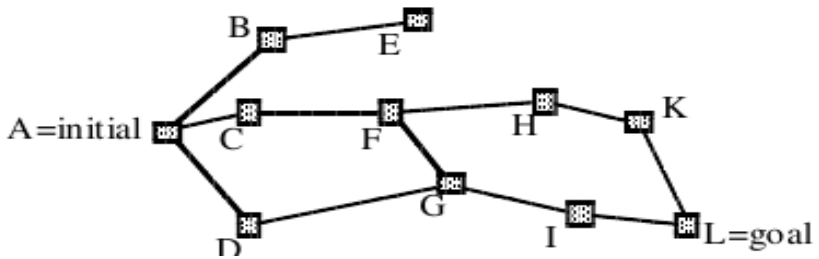
# Graph Search

- Methods
  - Breath First
  - Depth First
  - Dijkstra
  - A* and variants
  - D* and variants
  - ...
- Discriminators
  - $f(n) = g(n) + \varepsilon\, h(n)$
  - $g(n') = g(n) + c(n,n')$



# Graph Search Strategies: Breadth-First Search



# Graph Search Strategies: Breadth-First Search (cont.)

- Corresponds to a wavefront expansion on a 2D grid
- Use of a FIFO queue
  - First-found solution is optimal if all edges have equal costs
- Dijkstra's search is an „g(n)-sorted" HEAP variation of breadth first search
  - First-found solution is guaranteed to be optimal no matter the cell cost

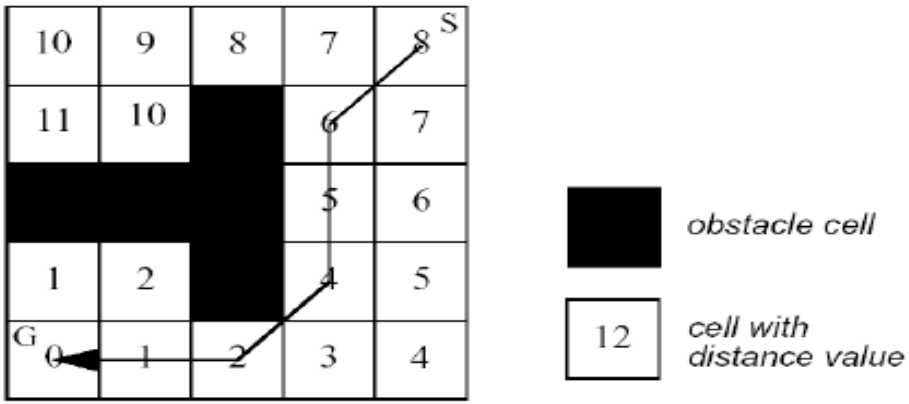
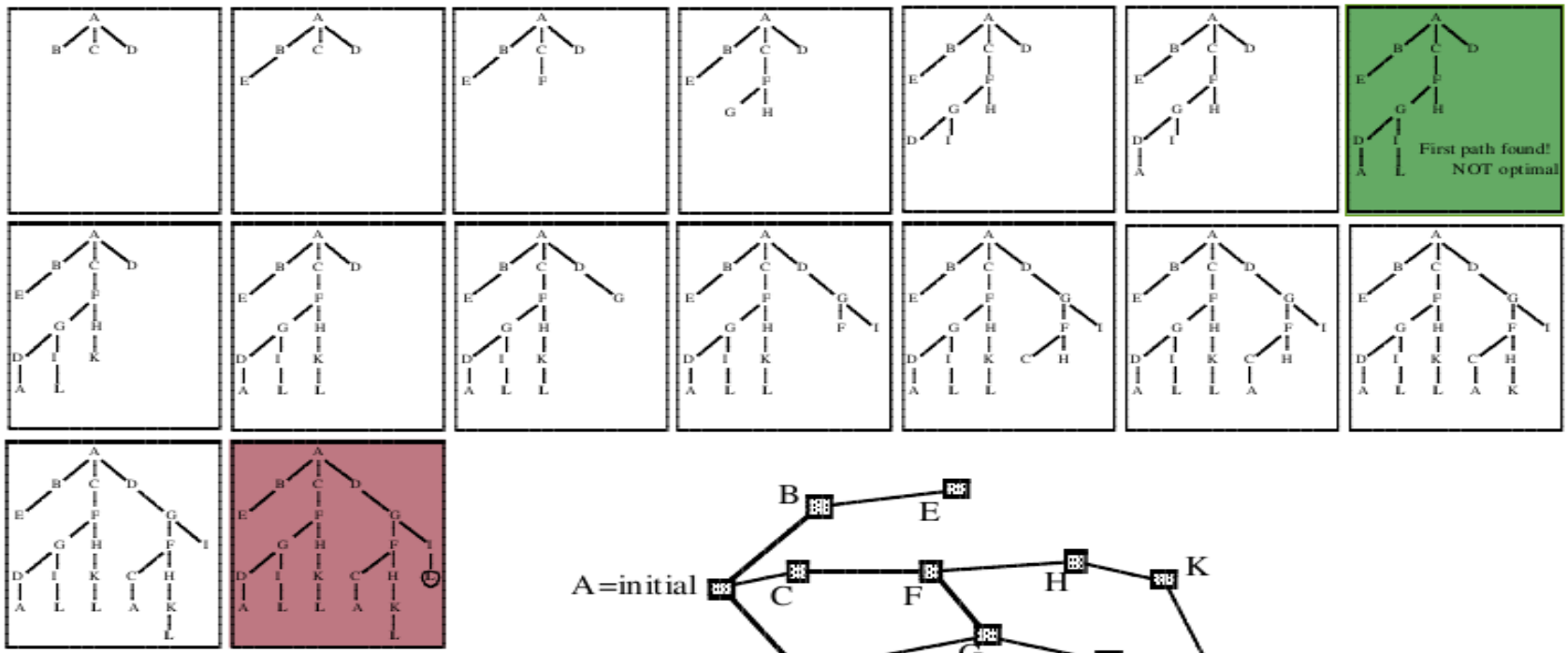



Fig. 1: NF1: put in each cell its $L^1$-distance from the goal position (used also in local path planning)

# Graph Search Strategies: Depth-First Search



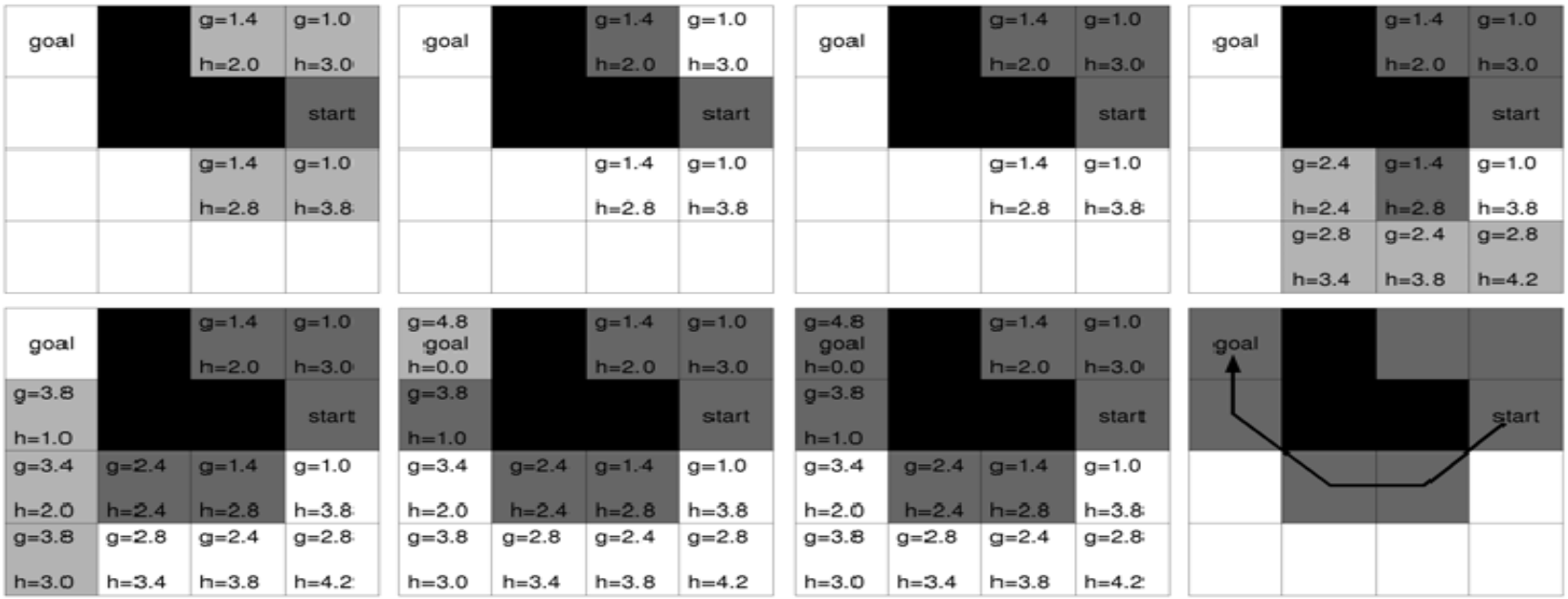




- Use of a LIFO queue
- Memory efficient (fully explored subtrees can be deleted)

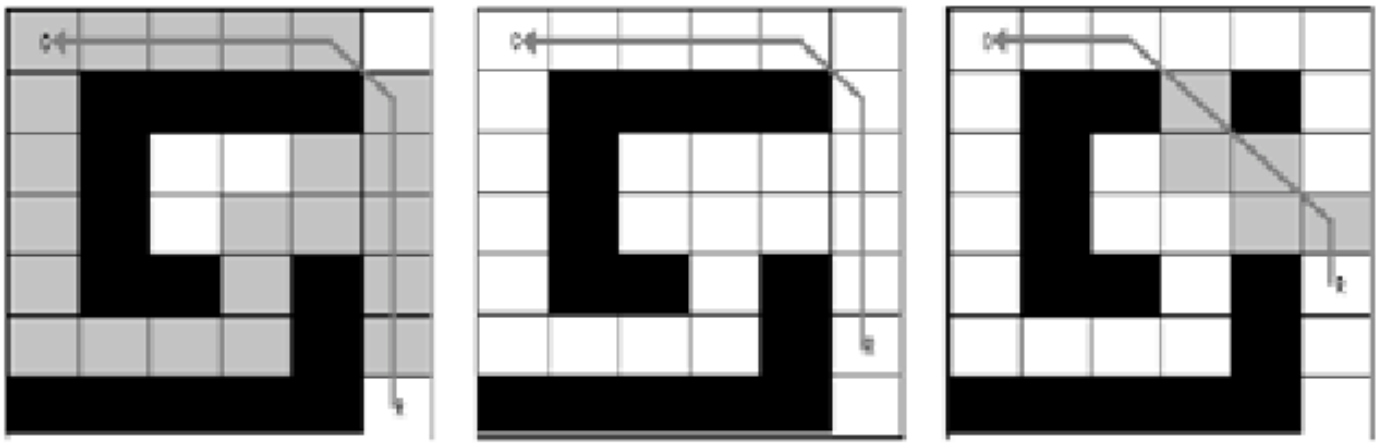
# Graph Search Strategies: A* Search

- Similar to Dijkstra's algorithm, A* also uses a HEAP (but „f(n)-sorted")
- A* uses a heuristic function h(n) (often euclidean distance)
- $f(n) = g(n) + \varepsilon\, h(n)$



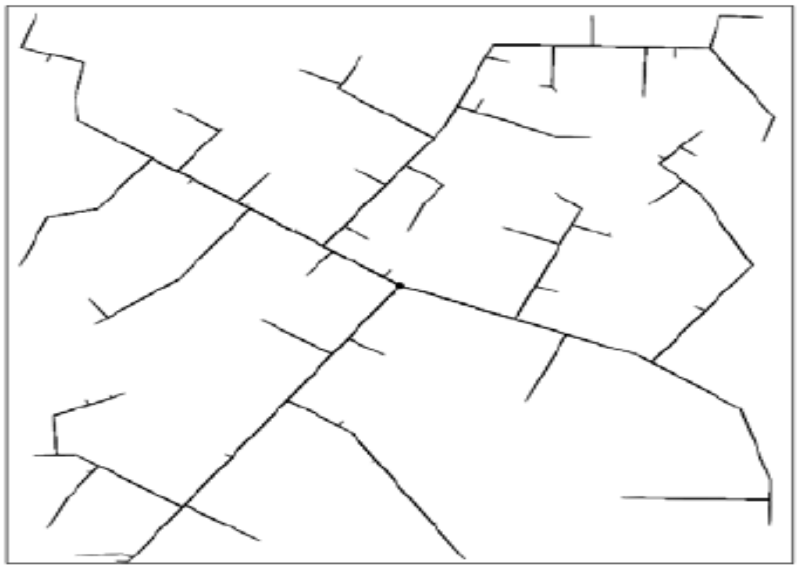
# Graph Search Strategies: D* Search

- Similar to A* search, except that the search begins from the goal outward
- $f(n) = g(n) + \varepsilon\, h(n)$
- First pass is identical to A*
- Subsequent passes reuse information from previous searches

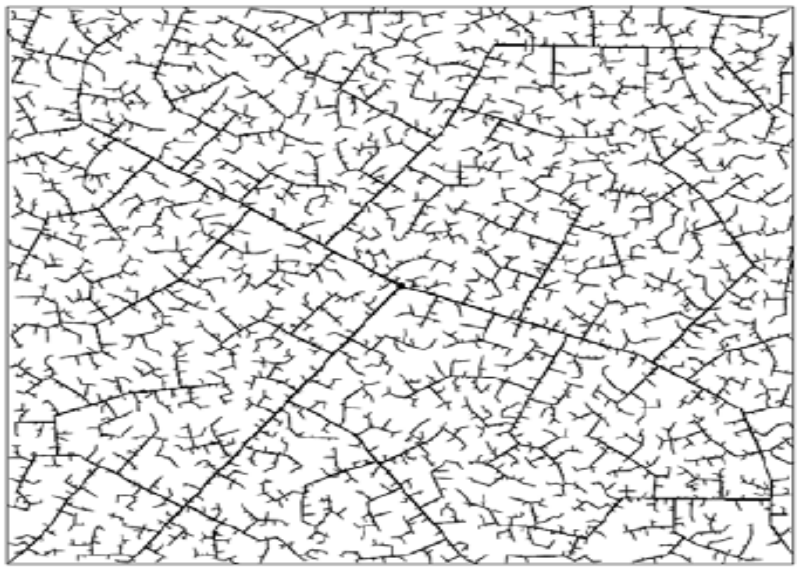


# Graph Search Strategies: Randomized Search

- Most popular version is the rapidly exploring random tree (RRT)
  - Well suited for high-dimensional search spaces
  - Often produces highly suboptimal solutions



45 iterations



2345 iterations